# ABSTRACT

Efficient memory usage is crucial for data-intensive applications as a smaller memory footprint ensures better cache performance and allows one to run a larger problem size given a fixed amount of main memory. The solutions found by existing techniques for automatic storage optimization for arrays in affine loop-nests, which minimize the storage requirements for the arrays, are often far from good or optimal and could even miss nearly all storage optimization potential. In this work, we present a new automatic storage optimization framework and techniques that can be used to achieve intra-array as well as inter-array storage reuse within affine loop-nests with a pre-determined schedule.

Over the last two decades, several heuristics have been developed for achieving complex transformations of affine loop-nests using the polyhedral model. However, there are no comparably strong heuristics for tackling the problem of automatic memory footprint optimization. We tackle the problem of storage optimization for arrays by formulating it as one of finding the right storage partitioning hyperplanes: each storage partition corresponds to a single storage location. Statement-wise storage partitioning hyperplanes are determined that partition a unified global array space so that values with overlapping live ranges are not mapped to the same partition. Our integrated heuristic for exploiting intra-array as well as inter-array reuse opportunities is driven by a fourfold objective function that not only minimizes the dimensionality and storage requirements of arrays required for each high-level statement, but also maximizes inter-statement storage reuse.

We built an automatic polyhedral storage optimizer called SMO using our storage partitioning approach. Storage reduction factors and other results we report from SMO demonstrate the effectiveness of our approach on several benchmarks drawn from the domains of image processing, stencil computations, high-performance computing, and the class of tiled codes in general. The reductions in storage requirement over previous approaches range from a constant factor to asymptotic in the loop blocking factor or array extents – the latter being a dramatic improvement for practical purposes.

As an incidental and related topic, we also studied the problem of polyhedral compilation of graphical dataow programs. While polyhedral techniques for program transformation are now used in several proprietary and open source compilers, most of the research on polyhedral compilation has focused on imperative languages such as C, where the computation is specied in terms of statements with zero or more nested loops and other control structures around them. Graphical dataflow languages, where there is no notion of statements or a schedule specifying their relative execution order, have so far not been studied using a powerful transformation or optimization approach. The execution semantics and referential transparency of dataflow languages impose a different set of challenges. In this work, we attempt to bridge this gap by presenting techniques that can be used to extract polyhedral representation from dataflow programs and to synthesize them from their equivalent polyhedral representation. We then describe PolyGLoT, a framework for automatic transformation of dataflow programs that we built using our techniques and other popular research tools such as Clan and Pluto. For the purpose of experimental evaluation, we used our tools to compile LabVIEW, one of the most widely used dataflow programming languages. Results show that dataflow programs transformed using our framework are able to outperform those compiled otherwise by up to a factor of seventeen, with a mean speed-up of 2.30× while running on an 8-core Intel system.